



# **Scaling Rails on App Engine with JRuby and Duby**

**Run your apps on Google Servers,  
with access to first-class Java APIs**

John Woodell  
David Masover  
Ryan Brown  
June 9, 2010



# Google App Engine

# Key Features

- No need to install or maintain your own stack
- Use Google scalable services via standard APIs
- Built-in application management console
- Pay-as-you-go, with free quota to get started



# Key Limitations

- No native code
- No threads or sockets
- No writing to the filesystem
- No more than 30 seconds per request



# Quotas and Billing

Resource	Provided Free	Additional Cost
CPU time	6.5 hours/day	\$0.10/hour
Bandwidth In	1GByte/day	\$0.10/GByte
Bandwidth Out	1GByte/day	\$0.12/GByte
Stored Data	1 GB	\$0.005/GB-day
Emails sent	2000/day to users 50000/day to admins	\$0.0001/email

# App Engine Product Roadmap

- SSL for third-party domains
- Background servers capable of running for longer than 30s
- Ability to reserve instances to reduce application loading overhead
- Ability to select different availability vs. latency options for Datastore
- Support for mapping operations across datasets
- Datastore dump and restore facility
- Raise request/response size limits for some APIs
- Improved monitoring and alerting of application serving
- Support for Browser Push (Comet) communication
- Built-in support for OAuth & OpenID



# JRuby on App Engine

# Benefits of JRuby

- Outperforms MRI in many cases... 2x to 10x
- Gem extensions written in Java (no more segfaults)
- A wealth of integration options and first-class Java APIs

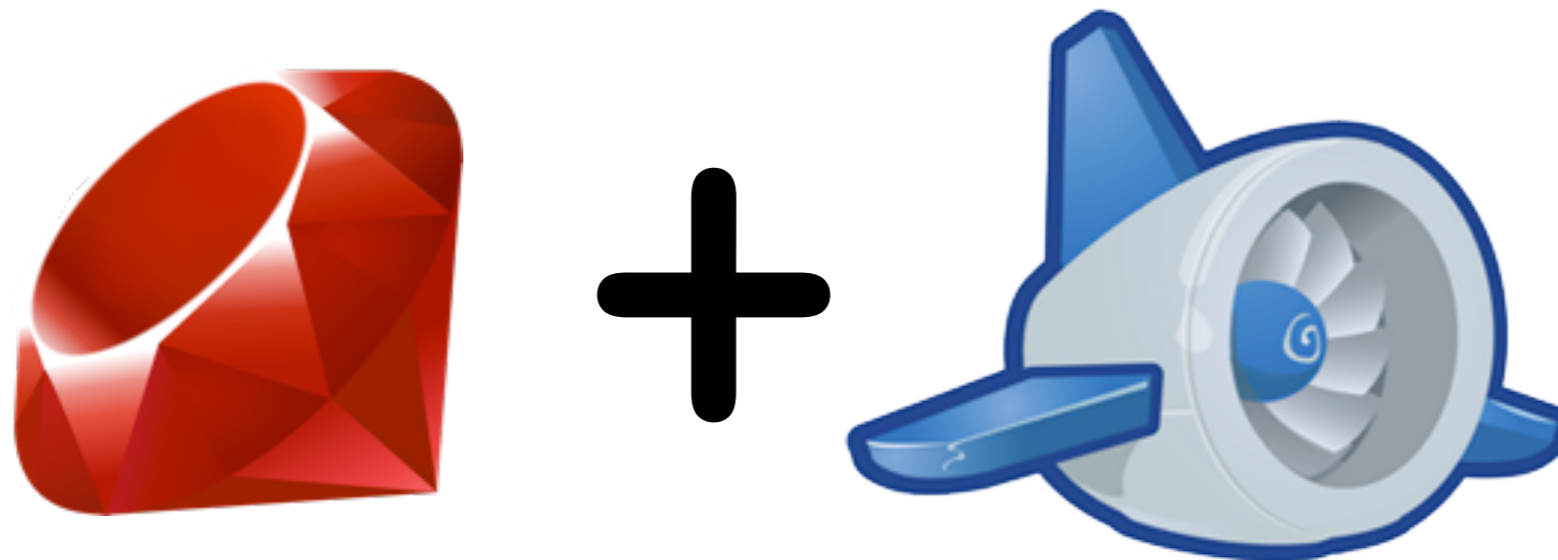


# App Engine JRuby Milestones

- 2009-04-08 @olabini publishes blog post on YARBL
- 2009-04-09 @nicksieger publishes warbler demo
- 2009-05-06 RailsConf (sinatra & merb)
- 2009-11-02 0.0.5 bundler, precompilation & Duby preview
- 2009-11-20 RubyConf (Rails 3.0.pre & Duby App)
- 2009-12-27 @urekat publishes Rails 2.3.5 patches
- 2010-01-11 @codingforrent published rails/dm gem
- 2010-01-21 0.0.8 Rails 2.3.5 Primer for DM & TinyDS
- 2010-01-26 0.0.9 Mechanize and Hpricot demos
- 2010-02-27 0.0.10 ActionMailer, ImageService, jruby-openssl
- 2010-04-08 @azazeal blog post on taxster.gr (OpenID)
- 2010-06-09 0.0.14 JRuby 1.5.1 & app.yaml preview

# Current Issues with JRuby on App Engine

- Several seconds to “spin-up” a new JRuby instance
- Some gems may need their extensions ported to Java
- Not officially supported, but you have all that you need



# Install it Now

```
sudo gem install google-appengine
```

Everything you need installs as gems

# App Engine Gems

- Development Gems
  - appengine-sdk
  - appengine-tools . . . dev\_appserver.rb & appcfg.rb
- Runtime Gems
  - appengine-rack . . . . jrubby-jars & jrubby-rack
  - appengine-apis
- Related Gems
  - dm-appengine
  - rails\_appengine
  - rails\_dm\_datastore
  - rails\_tiny\_ds

# App Engine JRuby APIs

- AppEngine::Users
- AppEngine::Datastore
- AppEngine::Memcache
- AppEngine::Mail
- AppEngine::URLFetch
- AppEngine::Images
- AppEngine::Logger
- AppEngine::XMPP
- AppEngine::Labs::TaskQueue
- AppEngine::OAuth
- AppEngine::Blobstore

# dm-appengine

David Masover

# DataMapper

```
class User
  property :id, Serial
  property :name, String
  property :email, String
end
```

```
User.create(:name => 'Nobody',
            :email => 'nobody@example.com')
```

```
User.first(:email => 'nobody@example.com')
```

```
User.auto_migrate! # on sqlite3, MySQL, etc
```

# ActiveRecord contains SQL strings

```
class Account < ActiveRecord::Base
  named_scope :rich,
    :conditions => 'balance >= 1000'
  named_scope :poor,
    :conditions => 'balance < 100'
end
```

```
class Account
  include DataMapper::Resource
  property :id, Serial
  property :balance, Integer

  def self.rich
    all(:balance.gte => 1000)
  end
  def self.poor
    all(:balance.lte => 100)
  end
end
```

# Use AppEngineResource

```
class User
  include DataMapper::AppEngineResource
  property :name, String
  property :email, Email
  ...
end
```

```
require 'dm-appengine/is_entity'
```

```
module DataMapper
  module AppEngineResource
    def self.included(klass)
      klass.class_eval do
        include DataMapper::Resource
        is :entity
      end
    end
  end
end
```

# Relationships

```
class User
  property :id, Key, :key => true

  property :ancestor, AncestorKey
  undef_method :ancestor
  undef_method :ancestor=

  belongs_to_entity :parent

def parent_id
  id.parent
end

def descendants(type)
  type.all(:ancestor => id)
end
end
```

# Simple

```
class User
  include DataMapper::AppEngineResource
  belongs_to entity :user
  has n, :children
end
```

# Transactions

- Idempotence
  - "Why can't I click submit more than once?"
- Transactions on datastore
  - Optimistic locking
  - Atomic
  - Only across entity-groups

# Simple case is Simple

```
class Post
  include DataMapper::AppEngineResource
  property :hit_count, Integer,
    :default => 0

  def hit!
    transaction do
      reload
      self.hit_count += 1
    save
  end
end
end
```

# Entity-groups

```
class User
  include DataMapper::AppEngineResource
  property :name, String
  has_descendants :favorites

  def set_favorite! name, url
    transaction do
      reload
      fave = self.favorites.first(:name => name)
      current ||= Fav.new(:id => {:parent_id => id})
      current.url = url
      current.save
    end
  end
end

class Fav
  property :name, String
  property :url, String
end
```

# Global transactions

- Don't do them!
- Really, you don't need them!
- They wouldn't scale anyway!

**but sometimes...**

# Inventory tracking

```
class Item
  include Base

  property :name, String, :required => true
  property :description, Text
  property :stock, Integer, :default => 0

  has_descendants :item_transactions
  ...
end
```

```
class ItemTransaction
  include Base

  property :count, Integer, :default => 1
  property :updated_at, DateTime

  belongs_to_entity :cart
end
```

# Shopping Cart

```
class Cart
  include Base

  property :state, String, :default => 'browse'
  property :updated_at, DateTime

  has_descendants :cart_items
  ...
end

class CartItem
  include Base

  property :count, Integer, :default => 1

  belongs_to_entity :item
end
```

# Duby

Ryan Brown

# Duby's not Ruby

- Statically typed
- No DUBY runtime
- Uses Java's type system

# Duby has Ruby-inspired Syntax

```
def fib(a:int)
  if a < 2
    a
  else
    fib(a - 1) + fib(a - 2)
  end
end

puts fib 10
```

```
// Generated from examples/Test.duby

public class Test extends java.lang.Object {

    public static void main(String[] argv) {
        System.out.println(Test.fib(10));
    }

    public static int fib(int a) {
        return (a < 2) ?
            (a) :
            ((Test.fib((a - 1)) + Test.fib((a - 2))));
    }
}
```

# A Simple Duby App

```
import javax.servlet.http.HttpServlet
import com.google.appengine.ext.duby.db.Model

class Post < Model
  property title, String
  property body, Text
end

class DubyApp < HttpServlet
  def_edb(list, 'com/ribrdb/list.dhtml')

  def doGet(request, response)
    @posts = Post.all.run
    response.getWriter.write(list)
  end

  def doPost(request, response)
    post = Post.new
    post.title = request.getParameter('title')
    post.body = Text.new(request.getParameter('body'))
    post.save
    doGet(request, response)
  end
end
```

# A Simple Duby App

```
import javax.servlet.http.HttpServlet
import com.google.appengine.ext.duby.db.Model

class Post < Model
  property title, String
  property body, Text
end

class DubyApp < HttpServlet
  def_edb(list, 'com/ribrdb/list.dht

  def doGet(request, response)
    @posts = Post.all.run
    response.getWriter.write(list)
  end

  def doPost(request, response)
    post = Post.new
    post.title = request.getParameter('title')
    post.body = Text.new(request.getParameter('body'))
    post.save
    doGet(request, response)
  end
end
```

Types  
inferred from parent  
class

# A Simple Duby App

```
import javax.servlet.http.HttpServlet
import com.google.appengine.ext.duby.db.Model

class Post < Model
  property title, String
  property body, Text
end

class DubyApp < HttpServlet
  def_edb(list, 'com/ribrdb/list.dhtml')

  def doGet(request, response)
    @posts = Post.all.run
    response.getWriter.write(list)
  end

  def doPost(request, response)
    post = Post.new
    post.title = request.getParameter('title')
    post.body = Text.new(request.getParameter('body'))
    post.save
    doGet(request, response)
  end
end
```



# Duby Supports Plugins

Only form of metaprogramming (for now)

```
require 'erb'
```

```
Duby::AST.defmacro('def_edb') do |duby, fcall, p|
  name = fcall.args_node.get(0).name
  path = fcall.args_node.get(1).value
  erb = ERB::Compiler.new(nil)
  ...
  src = erb.compile(IO.read(path))
  duby.eval(src, p, "(edb)")
end
```

# Duby Can Use erb Templates

```
<title>Posts</title>
<body>
  <h1>All Posts:</h1>
  <% for post in @posts %>
    <h2><%= post.title %></h2>
    <p><%= post.body.getValue %></p>
  <% end %>
  <hr>
  <h1>New Post:</h1>
  <form method=post>
    Title: <input type=text name=title><br>
    Body: <textarea name=body></textarea><br>
    <input type=submit>
  </form>
</body>
```

# More to Come

- Open classes
- Mix ins
- Macros
- More

# Resources

- Blog
  - <http://jrubby-appengine.blogspot.com/>
- Code Site
  - <http://code.google.com/p/appengine-jruby/>
- Google Group
  - <http://groups.google.com/group/appengine-jruby>
- Resources
  - <http://rails2010-dm.appspot.com/>
  - <http://github.com/masover/rails2010>
  - <http://github.com/masover/cart>
  - <http://cart-demo.appspot.com/>

